C2000 Interrupts and Peripheral Explorer Codec
Interrupts (1)

• Normally a program executes instructions in sequence.
• To react to an event (i.e., an A/D value) the processor must “poll” the device.
• This can be costly in terms of program execution.
• An alternative is to use “interrupts.”
Interrupts (2)

• When a hardware (or software) event occurs, it can trigger an “interrupt.”

• The processor stops whatever it is doing, saves important information, and immediately jumps to the “Interrupt Service Routine,” or ISR

• After the interrupt is serviced, the important information is restored, and the program resumes execution

• This is much more efficient than polling.

• We will use an interrupt from the external Codec (from the A/D converter) using McBSP (Multichannel Buffered Serial Port)
Interrupt Setup (1)

• Initialize Codec to generate interrupt on regular intervals (i.e., 44.1 kHz)
• Every time an interrupt occurs, do a task and then send data to D/A converters.
• Main routine doesn’t need to do anything

```c
//--- Main Loop
while(1) // endless loop
{
    // 1/2 sec delay (Max usec delay = 65535)
    for (i=0; i<10; i++) DelayUs(50000);   // DelayUs = delay microseconds
    GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;   // GPIO31 toggles
}
```
Interrupt Sources

Internal Sources
- TINT2
- TINT1
- TINT0

External Sources
- XINT1 - XINT3
- TZx
- XRS

ePWM, eCAP, eQEP, ADC, SCI, SPI, I2C, eCAN, McBSP, DMA, CLA, WD

PIE (Peripheral Interrupt Expansion)

F28x CORE
- XRS
- NMI
- INT1
- INT2
- INT3
- INT12
- INT13
- INT14
Maskable Interrupt Processing
Conceptual Core Overview

Core Interrupt  (IFR) “Latch”  (IER) “Switch”  (INTM) “Global Switch”

INT1  1

INT2  0

...  ...

INT14  1

F28x Core

- A valid signal on a specific interrupt line causes the latch to display a “1” in the appropriate bit
- If the individual and global switches are turned “on” the interrupt reaches the core

Note:
Peripheral Interrupt Expansion - PIE

PIE module for 96 Interrupts

INT1.x interrupt group
INT2.x interrupt group
INT3.x interrupt group
INT4.x interrupt group
INT5.x interrupt group
INT6.x interrupt group
INT7.x interrupt group
INT8.x interrupt group
INT9.x interrupt group
INT10.x interrupt group
INT11.x interrupt group
INT12.x interrupt group

Interrupt Group 1

PIEIFR1  PIEIER1

INT1.1 → 1
INT1.2 → 0
... → ...
INT1.8 → 1

INT1

28x Core Interrupt logic

IFR  IER  INTO  28x Core

INT1 – INT12

12 Interrupts

INT13 (TINT1)
INT14 (TINT2)
NMI
<table>
<thead>
<tr>
<th></th>
<th>INTx.8</th>
<th>INTx.7</th>
<th>INTx.6</th>
<th>INTx.5</th>
<th>INTx.4</th>
<th>INTx.3</th>
<th>INTx.2</th>
<th>INTx.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT1</td>
<td>WAKEINT</td>
<td>TINT0</td>
<td>ADCINT9</td>
<td>XINT2</td>
<td>XINT1</td>
<td>ADCINT2</td>
<td>ADCINT1</td>
<td></td>
</tr>
<tr>
<td>INT2</td>
<td>EPWM8_TZINT</td>
<td>EPWM7_TZINT</td>
<td>EPWM6_TZINT</td>
<td>EPWM5_TZINT</td>
<td>EPWM4_TZINT</td>
<td>EPWM3_TZINT</td>
<td>EPWM2_TZINT</td>
<td>EPWM1_TZINT</td>
</tr>
<tr>
<td>INT3</td>
<td>EPWM8_INT</td>
<td>EPWM7_INT</td>
<td>EPWM6_INT</td>
<td>EPWM5_INT</td>
<td>EPWM4_INT</td>
<td>EPWM3_INT</td>
<td>EPWM2_INT</td>
<td>EPWM1_INT</td>
</tr>
<tr>
<td>INT4</td>
<td>HRCAP2_INT</td>
<td>HRCAP1_INT</td>
<td></td>
<td></td>
<td></td>
<td>ECAP3_INT</td>
<td>ECAP2_INT</td>
<td>ECAP1_INT</td>
</tr>
<tr>
<td>INT5</td>
<td></td>
<td></td>
<td></td>
<td>HRCAP4_INT</td>
<td>HRCAP3_INT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I2CINT2A</td>
<td>I2CINT1A</td>
</tr>
<tr>
<td>INT9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- INT: Interrupt
- WAKEINT: Wakeup Interrupt
- TINT0: Timer Interrupt 0
- ADCINT9: ADC Interrupt 9
- XINT2: XINT2
- XINT1: XINT1
- ADCINT2: ADC Interrupt 2
- ADCINT1: ADC Interrupt 1
- EPWM8_TZINT: EPWM8_TZ Interrupt
- EPWM7_TZINT: EPWM7_TZ Interrupt
- EPWM6_TZINT: EPWM6_TZ Interrupt
- EPWM5_TZINT: EPWM5_TZ Interrupt
- EPWM4_TZINT: EPWM4_TZ Interrupt
- EPWM3_TZINT: EPWM3_TZ Interrupt
- EPWM2_TZINT: EPWM2_TZ Interrupt
- EPWM1_TZINT: EPWM1_TZ Interrupt
- EPWM8_INT: EPWM8_INT Interrupt
- EPWM7_INT: EPWM7_INT Interrupt
- EPWM6_INT: EPWM6_INT Interrupt
- EPWM5_INT: EPWM5_INT Interrupt
- EPWM4_INT: EPWM4_INT Interrupt
- EPWM3_INT: EPWM3_INT Interrupt
- EPWM2_INT: EPWM2_INT Interrupt
- EPWM1_INT: EPWM1_INT Interrupt
- HRCAP2_INT: HRCAP2_INT Interrupt
- HRCAP1_INT: HRCAP1_INT Interrupt
- HRCAP4_INT: HRCAP4_INT Interrupt
- HRCAP3_INT: HRCAP3_INT Interrupt
- ECAP3_INT: ECAP3_INT Interrupt
- ECAP2_INT: ECAP2_INT Interrupt
- ECAP1_INT: ECAP1_INT Interrupt
- MXINTA: Main External Interrupt A
- MRINTA: Main External Interrupt B
- DINTCH6: DINTCH6
- DINTCH5: DINTCH5
- DINTCH4: DINTCH4
- DINTCH3: DINTCH3
- DINTCH2: DINTCH2
- DINTCH1: DINTCH1
- I2CINT2A: I2C INT 2A
- I2CINT1A: I2C INT 1A
- ECAN1_INTA: ECAN1_INT A
- ECAN0_INTA: ECAN0_INT A
- SCITX_INTB: SCI Transmit INT B
- SCIRX_INTB: SCI Receive INT B
- SCITX_INTA: SCI Transmit INT A
- SCIRX_INTA: SCI Receive INT A
- ADCINT8: ADC Interrupt 8
- ADCINT7: ADC Interrupt 7
- ADCINT6: ADC Interrupt 6
- ADCINT5: ADC Interrupt 5
- ADCINT4: ADC Interrupt 4
- ADCINT3: ADC Interrupt 3
- ADCINT2: ADC Interrupt 2
- ADCINT1: ADC Interrupt 1
- CLA1_INT8: CLA1_INT 8
- CLA1_INT7: CLA1_INT 7
- CLA1_INT6: CLA1_INT 6
- CLA1_INT5: CLA1_INT 5
- CLA1_INT4: CLA1_INT 4
- CLA1_INT3: CLA1_INT 3
- CLA1_INT2: CLA1_INT 2
- CLA1_INT1: CLA1_INT 1
- LUF: Low Voltage Fault
- LVF: Low Voltage Fault
- XINT3: XINT3
Interrupt Setup (2)

In file: McBSPA.c

```c
//--- Enable the McBSP interrupt
PieCtrlRegs.PIEIER6.bit.INTx6 = 1; // Enable McBSP-A Transmit (bit 6) in PIE group 6
PieCtrlRegs.PIEIER6.bit.INTx5 = 1; // Enable McBSP-A Receive (bit 5) in PIE group 6
IER |= 0x0020; // Enable INT6 in IER to enable PIE group
```

In file: E71Shell.c

```c
//--- Enable global interrupts
asm(" CLRC INTM, DBGM"); // Enable global interrupts and realtime debug
```
Main Routine

In file: E71Shell.c

```c
//--- Global Variables
int adcLeft, adcRight;       // ADC left and right values
int dacLeft, dacRight;       // DAC left and right values
Uint16 sampleRate;           // Sampling rate

void main(void)
{
    int even = 0;
    unsigned int i;

    //Initialization
    InitSysCtrl();           // Initialize the CPU (FILE: SysCtrl.c)
    InitGpio();              // Initialize the shared GPIO pins (FILE: Gpio.c)
    InitPieCtrl();           // Initialize and enable the PIE (FILE: PieCtrl.c)
    InitWatchdog();          // Initialize the Watchdog Timer (FILE: WatchDog.c)
    InitSPIA();              // Initialize the SPI-A (FILE: SPI.c)
    InitMcBSPA();            // Initialize the McBSP-A – Including Interrupts(FILE: McBSP.c)

    InitAIC23();             // Initialize AIC23 codec on board.
    sampleRate = 44100;      // Initialize Sampling rate.

    asm(" CLRC INTM, DBGM"); // Enable global interrupts and realtime debug

    while(1)                 // Endless loop - but interrupts still occur.
    {
        asm(" NOP");       // Main routine doesn’t need to do anything.
    }
}
```
On Receive Interrupt from McBSP (A/D Values)

In file: DefaultISR.c

interrupt void MRINTA_ISR(void)            // PIE6.5 @ 0x000D98  MRINTA (McBSP-A)
{
    static volatile Uint16 cnt = 0;          // Counter for pin toggle
    GpioDataRegs.AI0DAT.bit.AI04 = 1;        // Set AI04 to 1 at beginning of ISR to time ISR

    // Increment count, if it is more than half the sample rate, toggle the pin.
    if (cnt++ >= (sampleRate>>1)) {
        GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1; // Toggle the pin
        cnt=0;
    }

    adcLeft = McbspaRegs.DRR2.all;           // Load left adc value
    adcRight = McbspaRegs.DRR1.all;          // Load right adc value
    E71Func();                               // Call function in E71Shell.c that does processing.
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;  // Must acknowledge the PIE group
    GpioDataRegs.AI0DAT.bit.AI04 = 0;        // Set AI04 to 0 at end of ISR
}
On Transmit Interrupt from McBSP (D/A Values)

In file: DefaultISR.c

```c
interrupt void MXINTA_ISR(void) // PIE6.6 @ 0x000D9A  MXINTA (McBSP-A)
{
    McbspaRegs.DXR2.all = dacLeft; // DAC left channel
    McbspaRegs.DXR1.all = dacRight; // DAC right channel

    PieCtrlRegs.PIEACK.all = PIEACK_GROUP6; // Must acknowledge the PIE group
}
```
E71Func()

// This function is called whenever the AIC does an ADC conversion.
// The left and right A/D value are in adcLeft and adcRight
// The *only* thing done in this function is to calculate new values
// for the DAC, and this should be done as efficiently as possible.
// The left and right D/A values should be put in dacLeft and dacRight.

void E71Func(void)
{
    // Swap left and right channels.
    dacLeft = adcRight;
    dacRight = adcLeft;
}